Design Patterns in Python

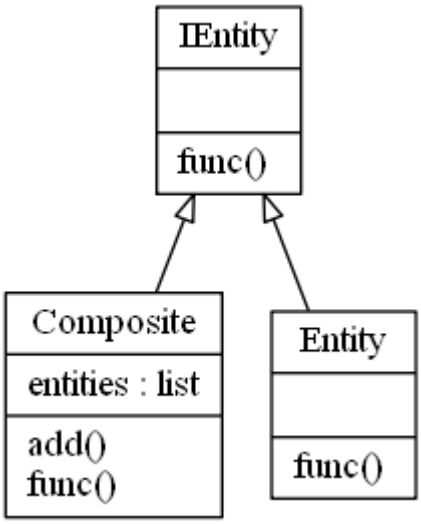
Composite Design Pattern

Description

The Composite design pattern,

- allows you to represent individual entities and groups of entities in the same manner.
- is a structural design pattern that lets you compose objects into a tree.
- is great if you need the option of swapping hierarchal relationships around.
- makes it easier for you to add new kinds of components
- provides flexibility of structure
- conform to the Single Responsibility Principle in the way that it separates the aggregation of objects from the features of the object.

Examples of using the Composite Design Pattern can be seen in a filesystem directory structure, where you ca n swap the hierarchy of folders, and in a drawing program where you can group, un-group and transform objects, and multiple objects at the same time.



Source Code

composite.py

```
from abc import ABCMeta, abstractmethod

class IGraphic(metaclass=ABCMeta):
    @staticmethod
    @abstractmethod
    def print():
        """print information"""

class Ellipse(IGraphic):
    def print(self):
        print("Ellipse")
```

```
class Circle(IGraphic):
    def print(self):
        print("Circle")

class CompositeGraphic(IGraphic):
    def __init__(self):
        self.child_graphics = []

    def add(self, graphic):
        self.child_graphics.append(graphic)

    def print(self):
        for g in self.child_graphics:
            g.print()


ELLIPSE1 = Ellipse()
CIRCLE1 = Circle()

COMPOSITE1 = CompositeGraphic()
COMPOSITE1.add(ELLIPSE1)

COMPOSITE2 = CompositeGraphic()
COMPOSITE2.add(CIRCLE1)
COMPOSITE2.add(COMPOSITE1)

COMPOSITE2.print()

# ELLIPSE1.print()
# CIRCLE1.print()
```